APIs for Persistent Memory Programming

MSST 2018

Andy Rudoff

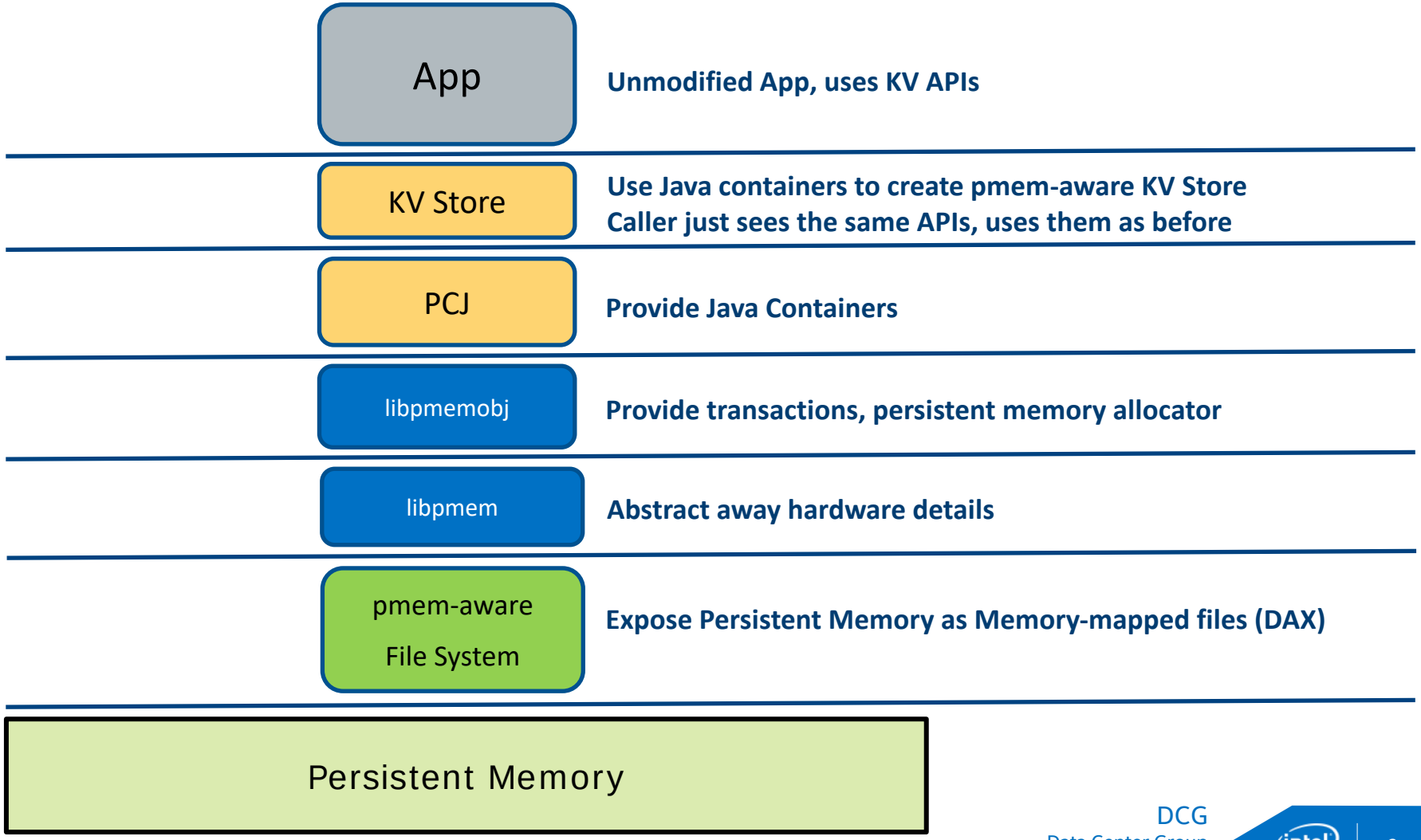
NVM Software Architect

Intel Corporation

DCG
Data Center Group

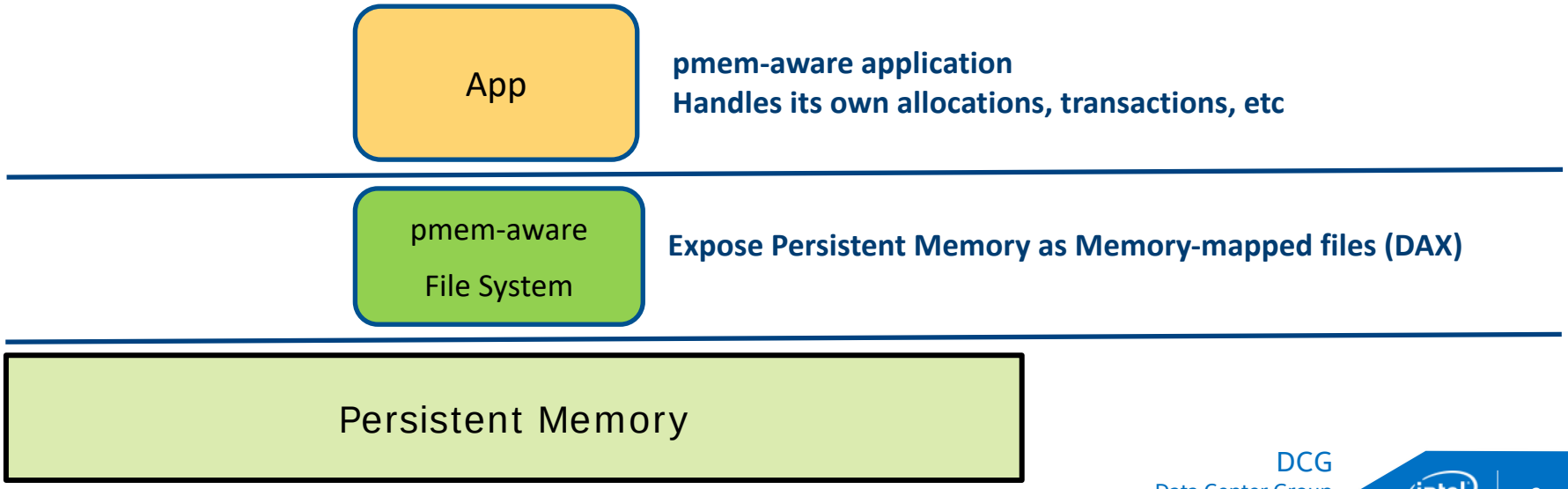
A Full-Stack Example

Using a key-value store as an example

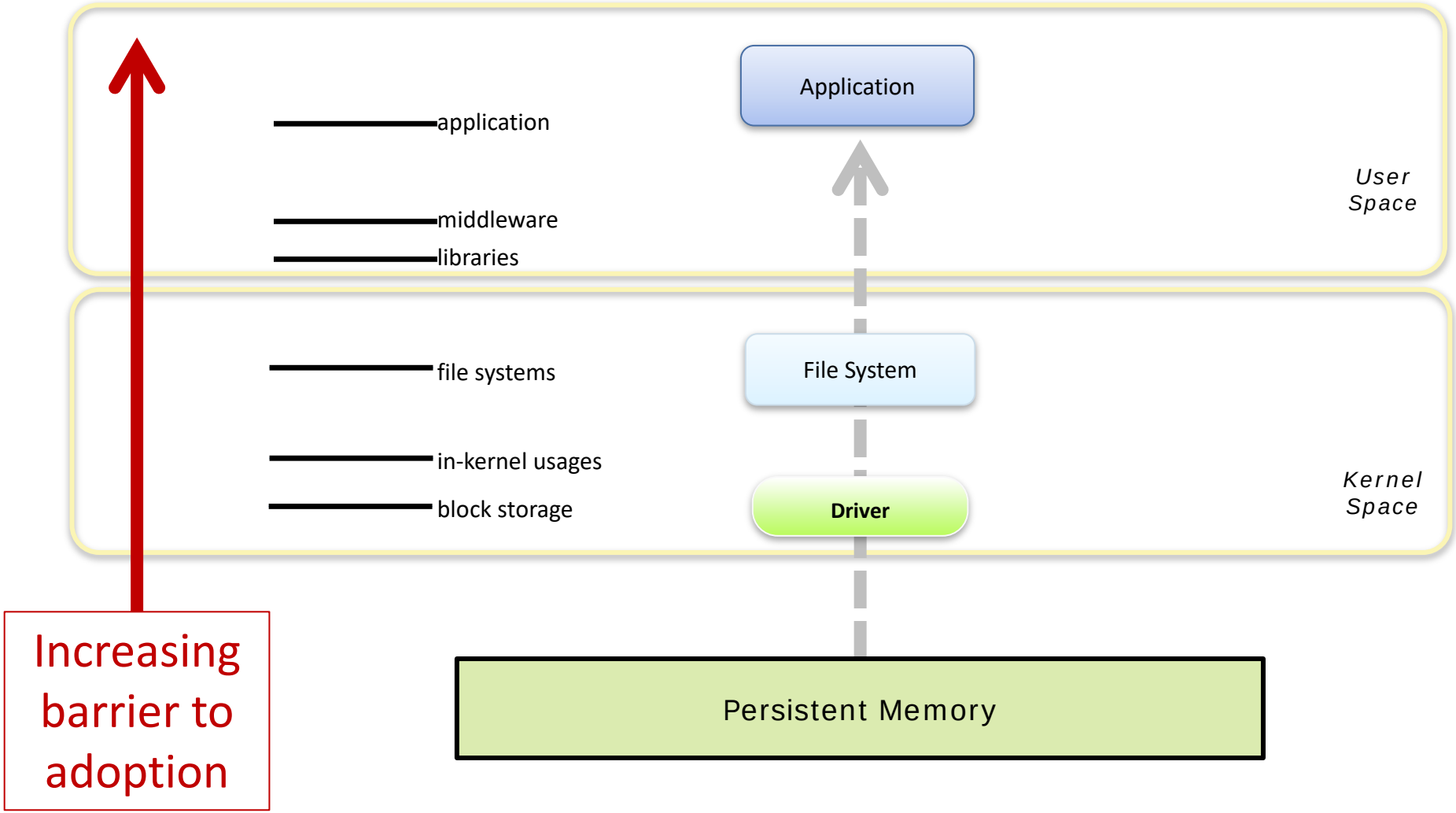


Another Full-Stack Example

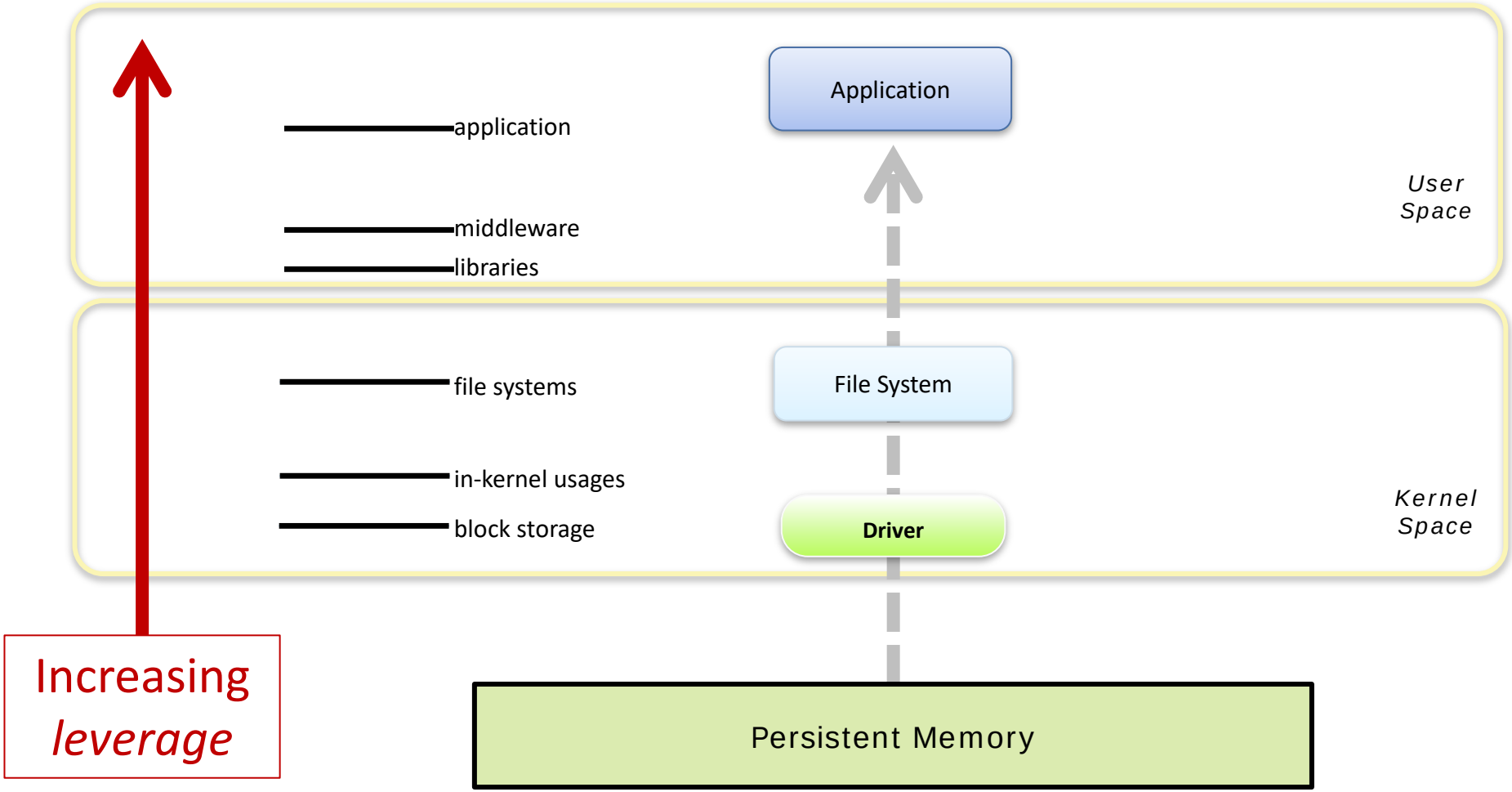
The app does everything



Transparency Levels for pmem



Transparency Levels for pmem



Ancient History

June 2012

- Formed the NVM Programming TWG
- Immediate participation from key OSVs, ISVs, IHVs

January 2013

- Held the first PM Summit (actually called “NVM Summit”)

January 2014

- TWG published rev 1.0 of the NVM Programming Model

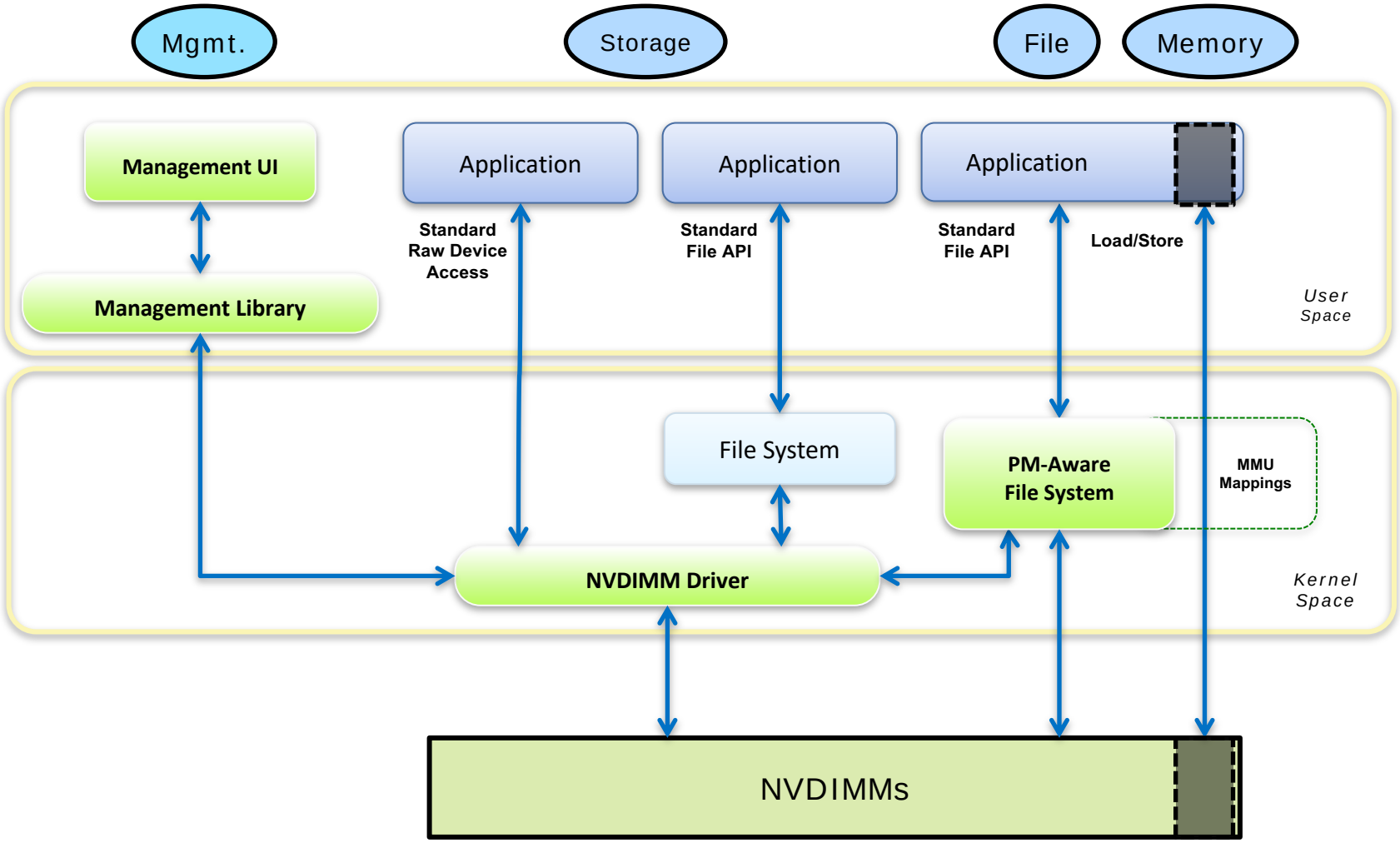
June 2017

- Rev 1.2 published

And now…

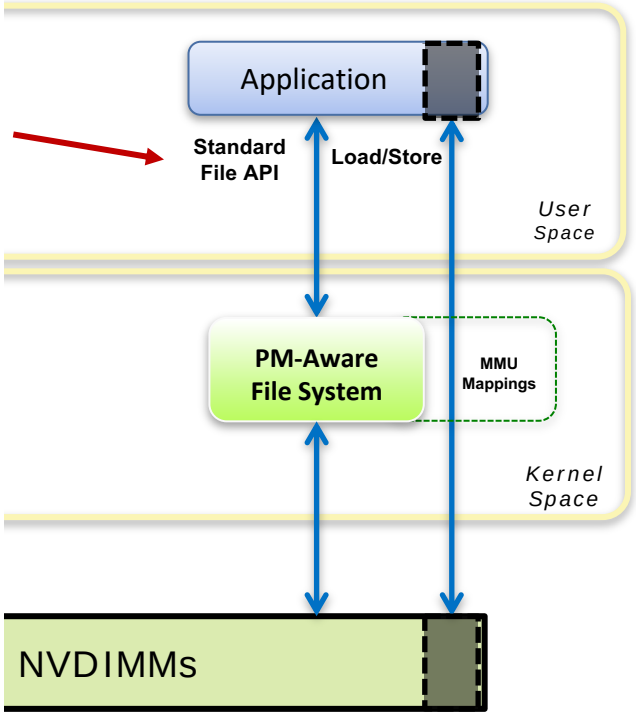
- Programming model supported & shipping in multiple operating systems
- APIs built on top of the programming model available

The SNIA NVM Programming Model



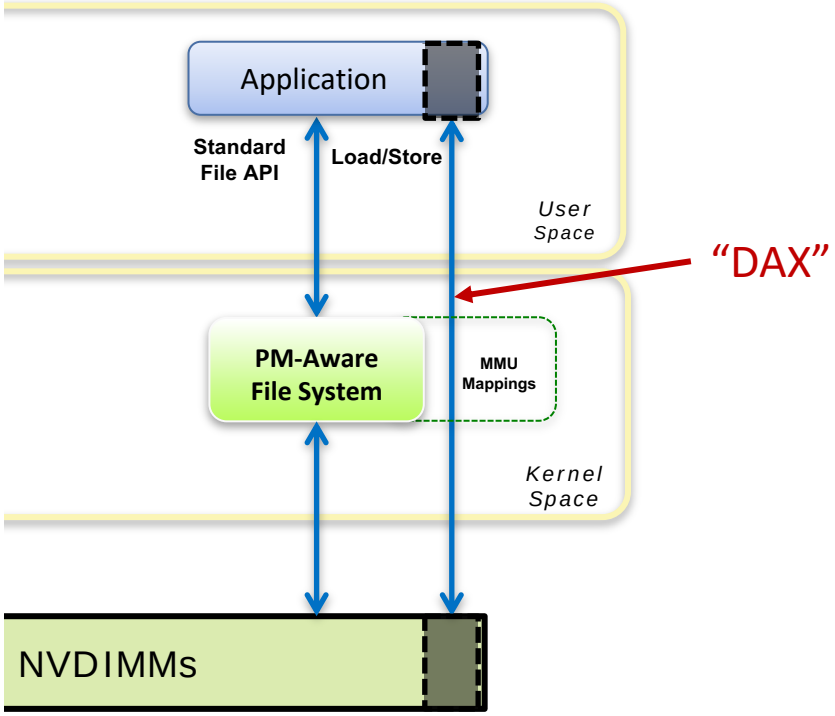
Must Open File Before Mapping

Standard Naming and Permission Model

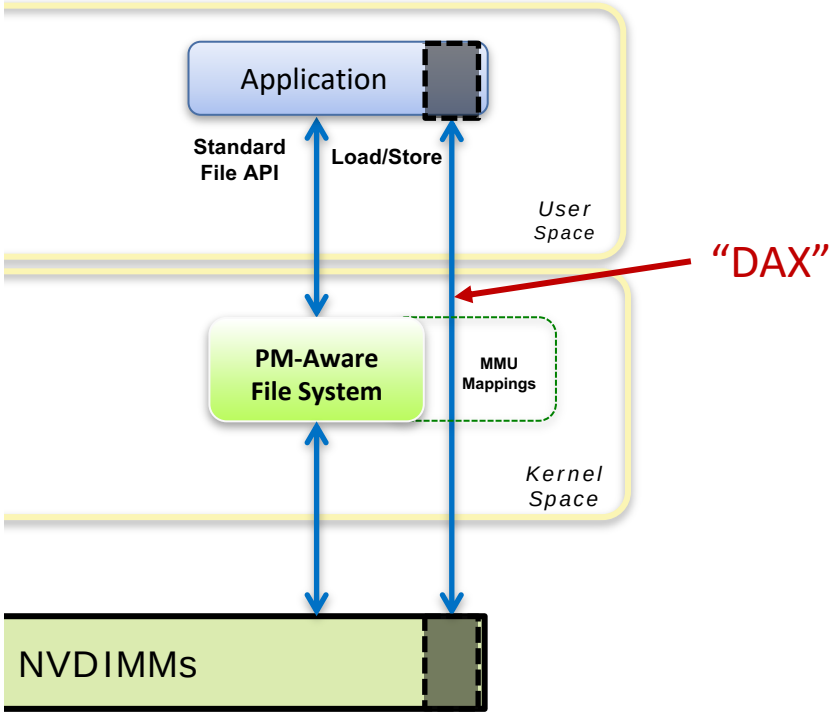


Direct Access

Definition: no paging, no page cache use



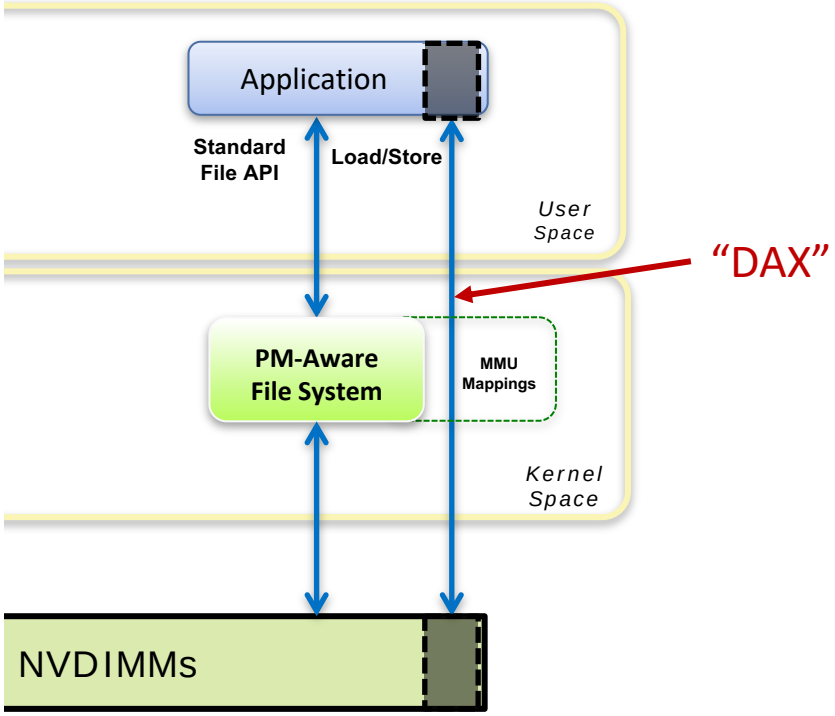
Direct Access



Windows:

- DAX Support is shipping
- NTFS is PM-Aware
- Some new APIs
- PMDK support

Direct Access

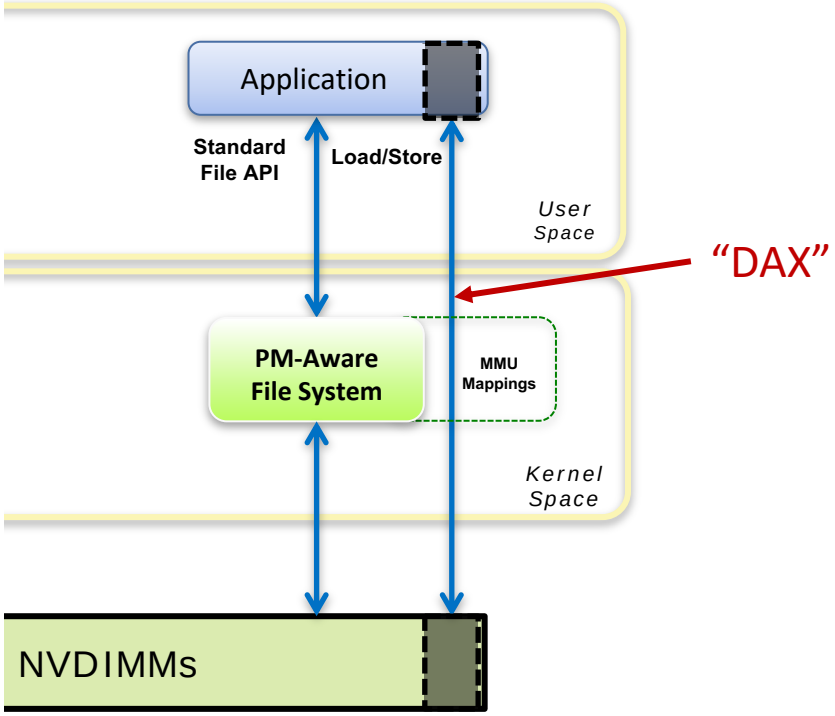


Linux:

- DAX Support is shipping
- ext4 is PM-Aware
- XFS is PM-Aware
- PMDK support

- More filesystems coming

Direct Access



Virtualization of PM:
- VMware
- Hyper-V
- KVM
- Xen

Applications: Public Demos

- 2017 was an interesting year for demos…

SAP SAPPHIRE

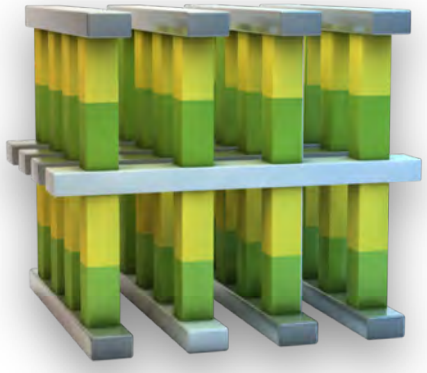
Oracle OpenWorld

- Built on the Persistent Memory programming model!

Intel Persistent Memory

New Type of Memory

- Persistent, Large Capacity & Byte Addressable
  - 6 TB per two-socket system
- DDR4 Socket Compatible
  - Can Co-exist with Conventional DDR4 DRAM DIMMs
- Cheaper than DRAM
- Availability
  - 2018



trivial.c

```
fd = open(filename, O_RDWR);

pmaddr = mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);

strcpy(pmaddr, "Hello, Persistent Memory!");

msync((void *)pmaddr, 4096, MS_SYNC);
```

trivial.c

```
fd = open(filename, O_RDWR);

pmaddr = mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

close(fd);

strcpy(pmaddr, "Hello, Persistent Memory!");

msync((void *)pmaddr, 4096, MS_SYNC);
```

- pmaddr could point to a really huge capacity – terabytes!
  - Want some allocator like malloc/free/new or language integration
- strcpy is not atomic
- msync is not atomic

- Basic memory-mapped files are not transactional – up to the caller

Also, you should know…

msync now just flushes CPU caches

- no page cache with DAX

Platforms may support "Optimized Flush"

- Flush changes from using user space instructions for performance
- Windows supports this
- Linux supports this with new MAP_SYNC flag

Future platforms may have persistent CPU caches

- ACPI property tells you this
- Write future-proof code by looking at this property & skipping flushes

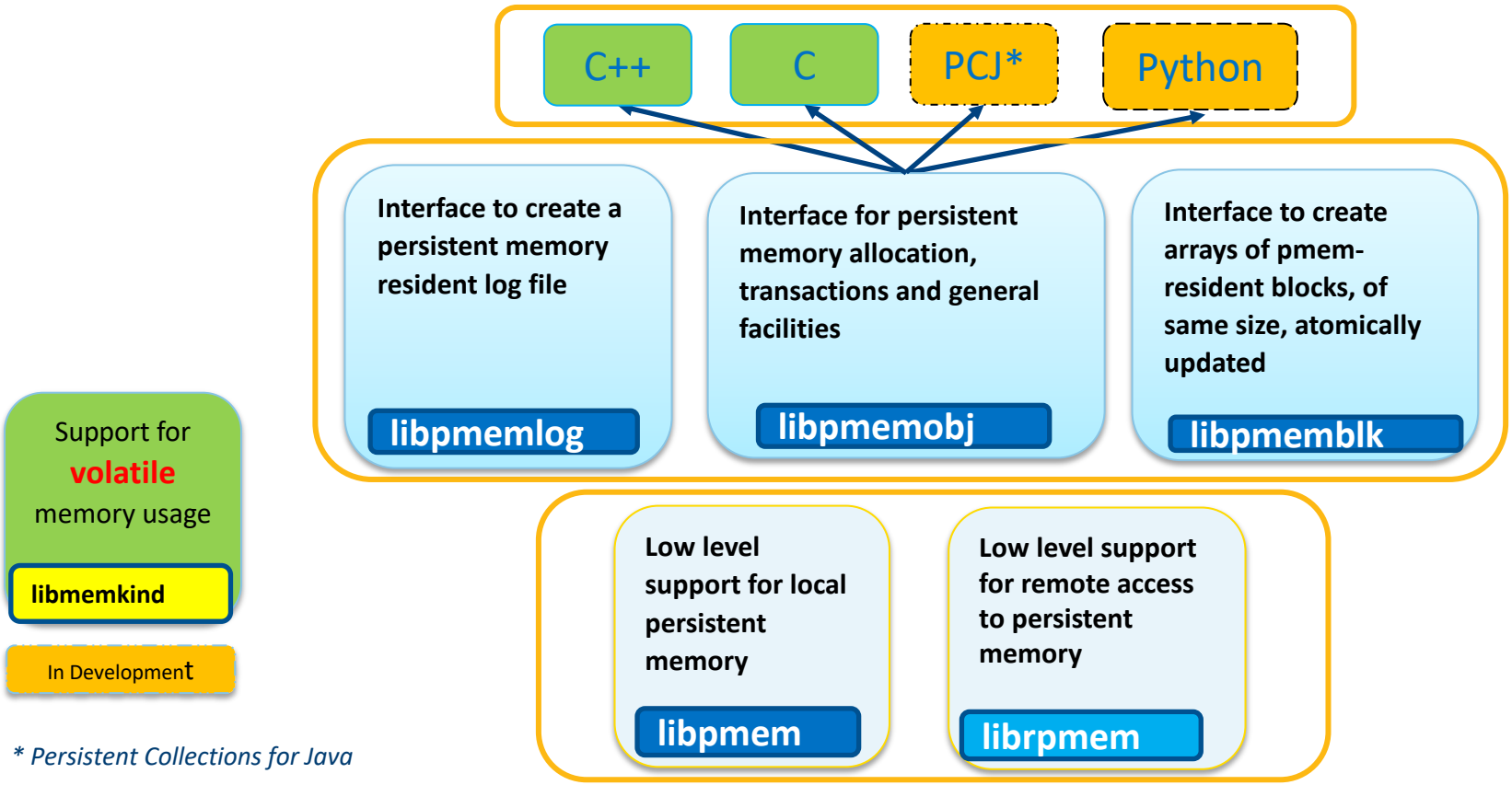
Persistent Memory errors appear as memory errors

- For example: SIGBUS on an uncorrectable in Linux

The Persistent Memory Development Kit
PMDK http://pmem.io

- PMDK is a collection of libraries
  - Developers pull only what they need
    - Low level programming support
    - Transaction APIs
  - Fully validated
  - Performance tuned.
- Open Source & Product neutral

PMDK Libraries



** Persistent Collections for Java*

Also, you should know…

PMDK libraries are validated to product quality

- Many hundreds of unit tests
- Many hundreds of system tests

We don't think we're all done now…

- Performance work continues, with some significant results
- Feature development continues
  - RAS
  - More mature language integration (especially around C++)
  - More mature replication

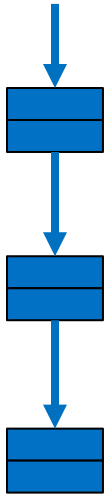
Still adding libraries

- Example: libpmemkv

PMDK in a Nutshell

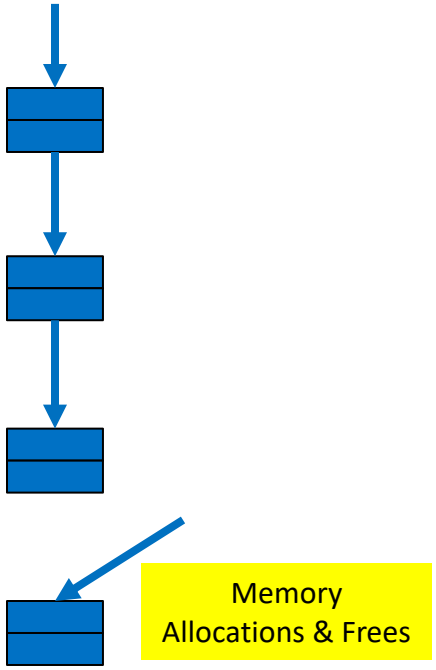
Ten libraries, tools, examples…

```
void push(pool_base &pop, uint64_t value) {
    transaction::exec_tx(pop, [&] {
        auto n = make_persistent<pmem_entry>();

        n->value = value;
        n->next = nullptr;
        if (head == nullptr) {
            head = tail = n;
        } else {
            tail->next = n;
            tail = n;
        }
    });
}
```



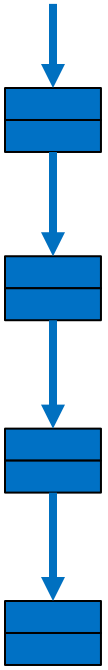
PMDK in a Nutshell

```
void push(pool_base &pop, uint64_t value) {
    transaction::exec_tx(pop, [&] {
        auto n = make_persistent<pmem_entry>();

        n->value = value;
        n->next = nullptr;
        if (head == nullptr) {
            head = tail = n;
        } else {
            tail->next = n;
            tail = n;
        }
    });
}
```



PMDK in a Nutshell

```
void push(pool_base &pop, uint64_t value) {
    transaction::exec_tx(pop, [&] {
        auto n = make_persistent<pmem_entry>();

        n->value = value;
        n->next = nullptr;
        if (head == nullptr) {
            head = tail = n;
        } else {
            tail->next = n;
            tail = n;
        }
    });
}
```



Multiple Operations Made Atomic

PMDK in a Nutshell

Complex transactions, allocation handled by libraries

- No “flush” calls to manage in most cases
- Each ISV doesn’t have to re-invent
- Performance tuned (esp for future enhancements)

Licensing is very liberal

- Steal all the code you want!

PMDK is a convenience, not a requirement

- Build your own library if you like!

Persistent Collections for Java

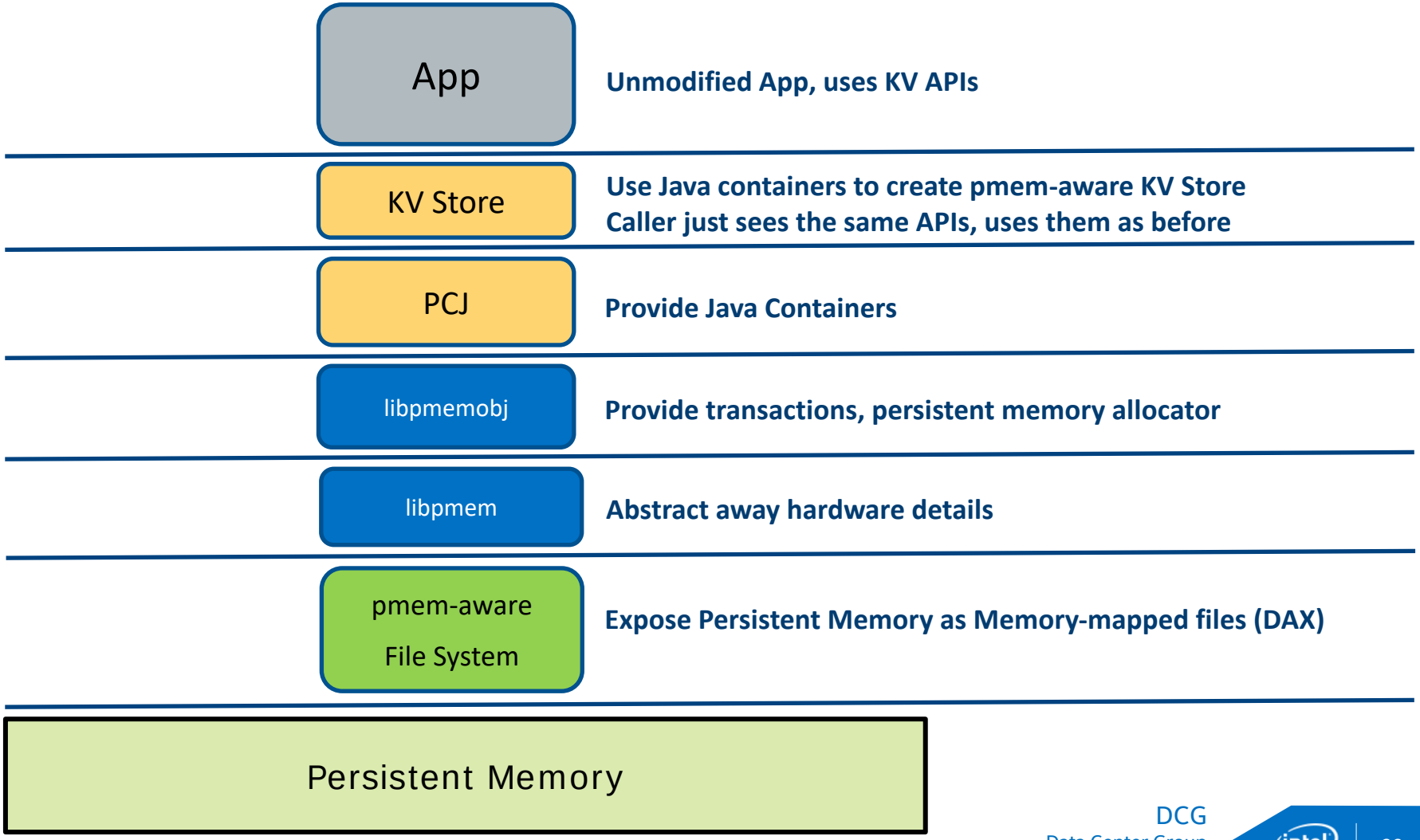
```
...
PersistentIntArray data = new PersistentIntArray(1024);
ObjectDirectory.put("My_fancy_persistent_array", data);   // no serialization
data.set(0, 123);
...
```

No flush calls.
Transactional.
Java library handles it all.

See “pilot” project at: https://github.com/pmem/pcj

A Full-Stack Example

Using a key-value store as an example



SNIA TWG Ongoing Work

Security

- PM Hardware Security Threat Model (balloting)

Remote persistent memory (via RDMA)

- Ongoing – optimizations for RDMA worked in multiple forums
- Remote asynchronous flush (under discussion)

Higher-level Semantics

- As we learn more..

More Information

http://snia.org/PM

- Specs, workgroups, webcasts, videos, presentations

http://pmem.io

- PMDK and other persistent memory programming information

http://pmem.io/documents

- Links to publications, standards, Windows & Linux info

http://software.intel.com/pmem

- Intel Developer Zone for persistent memory programming